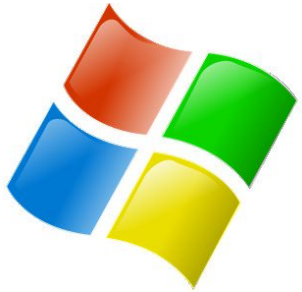
Corso di STATISTICA, INFORMATICA, ELABORAZIONE DELLE INFORMAZIONI

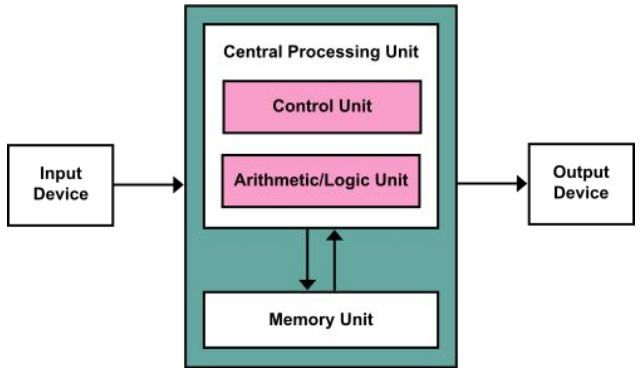
UNIVERSITÀ DEGLI STUDI DELLA BASILICATA

Modulo di Sistemi di Elaborazione delle Informazioni

Classi

Docente:
Domenico Daniele Bloisi

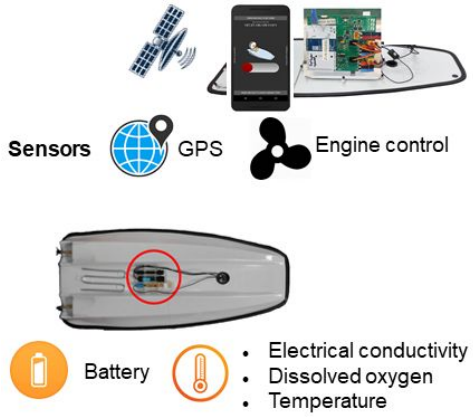


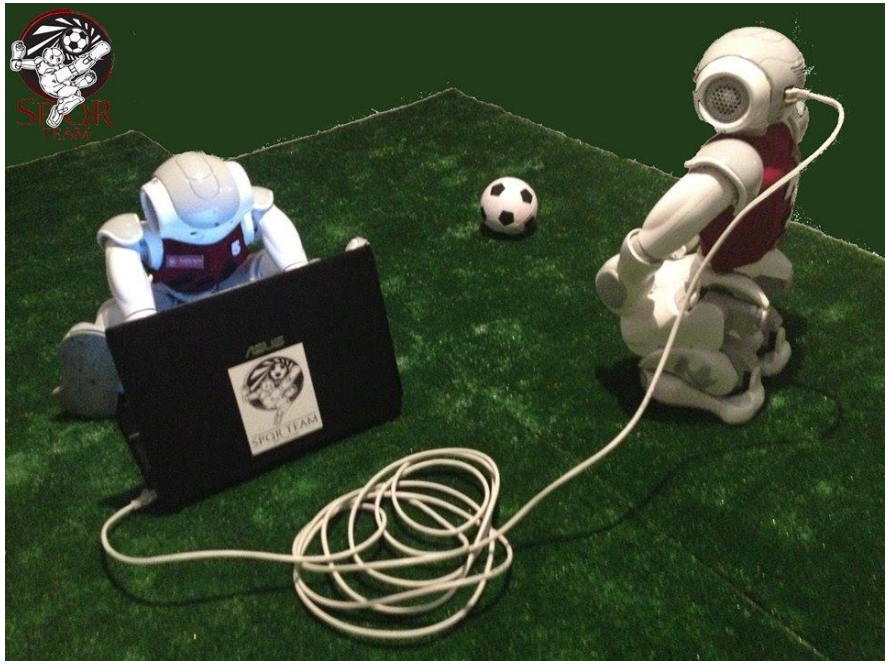


Domenico Daniele Bloisi

- Professore Associato
  Dipartimento di Matematica, Informatica ed Economia
  Università degli studi della Basilicata
  http://web.unibas.it/bloisi

- SPQR Robot Soccer Team
  Dipartimento di Informatica, Automatica e Gestionale Università degli studi di Roma “La Sapienza”
  http://spqr.diag.uniroma1.it



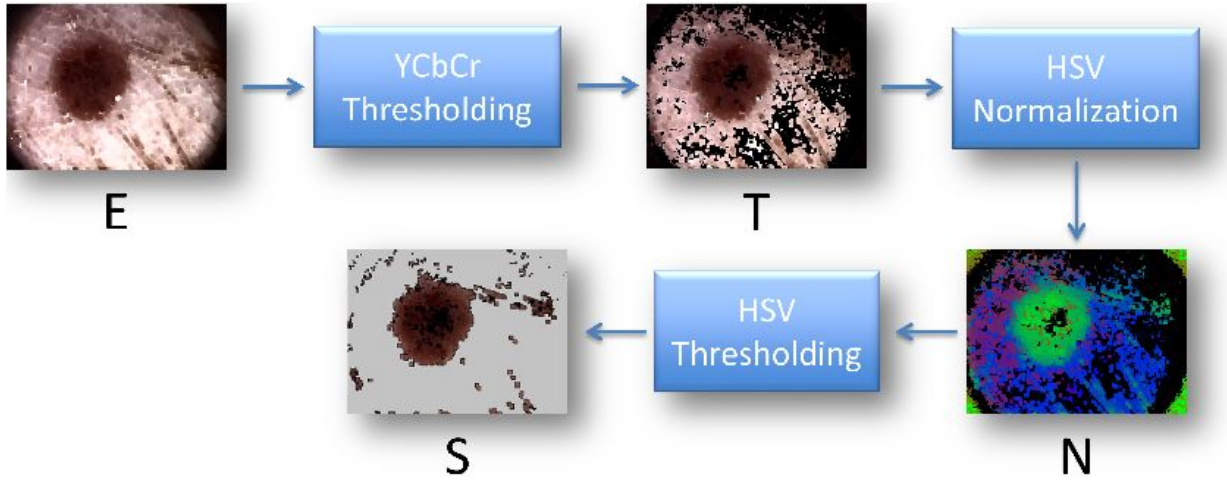


Interessi di ricerca

- Intelligent surveillance
- Robot vision
- Medical image analysis

https://youtu.be/2KHNZX7UIWQ

https://youtu.be/9a70Ucgbi_U



UNIBAS Wolves https://sites.google.com/unibas.it/wolves

- UNIBAS WOLVES is the robot soccer team of the University of Basilicata. Established in 2019, it is focussed on developing software for NAO soccer robots participating in RoboCup competitions.

- UNIBAS WOLVES team is twinned with SPQR Team at Sapienza University of Rome

https://youtu.be/ji0OmkaWh20

Informazioni sul corso

Il corso di STATISTICA, INFORMATICA, ELABORAZIONE DELLE INFORMAZIONI

- include 3 moduli:
  - SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI
    (il martedì - docente: Domenico Bloisi)
  - INFORMATICA
    (il mercoledì - docente: Enzo Veltri)
  - PROBABILITA' E STATISTICA MATEMATICA
    (il giovedì - docente: Antonella Iuliano)
- Periodo: I semestre ottobre 2022 – gennaio 2023

Ricevimento Bloisi

- In presenza, durante il periodo delle lezioni:
  Lunedì dalle 17:00 alle 18:00
  presso Edificio 3D, II piano, stanza 15
  Si invitano gli studenti a controllare regolarmente la bacheca degli avvisi per eventuali variazioni
- Tramite google Meet e al di fuori del periodo delle lezioni:
  da concordare con il docente tramite email

Per prenotare un appuntamento inviare una email a
domenico.bloisi@unibas.it

Recap

Dictionaries

```
rubrica = {"Antonio":"323573", "Giuseppe":"322955", "Marina":"3449007"}
```

Retrieving a Value from a Dictionary

```
[1] rubrica = {"Antonio":"323573", "Giuseppe":"322955", "Marina":"3449007"}
```

```
[2] rubrica
```

```
{'Antonio': '323573', 'Giuseppe': '322955', 'Marina': '3449007'}
```

```
if "Laura" in rubrica:
  print("Laura c'è")
else:
  print("Laura non c'è")
```

```
Laura non c'è
```

Retrieving a Value from a Dictionary

```
[11] print(rubrica["Marina"])

     3449007
```

```
[5]  print(rubrica[2])

     ---------------------------------------------------------------------------
     KeyError                                  Traceback (most recent call last)
     <ipython-input-5-1443c816d35a> in <module>
     ----> 1 print(rubrica[2])

     KeyError: 2

     SEARCH STACK OVERFLOW
```

```
     print(rubrica["Mario"])

     ---------------------------------------------------------------------------
     KeyError                                  Traceback (most recent call last)
     <ipython-input-12-6526382855a0> in <module>
     ----> 1 print(rubrica["Mario"])

     KeyError: 'Mario'

     SEARCH STACK OVERFLOW
```

Retrieving a Value from a Dictionary

```
[12] print(rubrica["Mario"])
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-12-6526382855a0> in <module>
----> 1 print(rubrica["Mario"])

KeyError: 'Mario'
```

SEARCH STACK OVERFLOW

```
if "Mario" not in rubrica:
  print("Mario non si trova.")
  print("Lo vuoi aggiungere?")
```

```
Mario non si trova.
Lo vuoi aggiungere?
```

Getting the Number of Elements and Mixing Data Types

```
[23] rubrica["Ethan"] = ["323500","336599"]
```

```
rubrica
```

```
{'Antonio': '322111',
 'Giuseppe': '322955',
 'Marina': '3449007',
 'Mario': '392356',
 'Ethan': ['323500', '336599']}
```

Creating an Empty Dictionary and Using `for` Loop to Iterate Over a Dictionary

```
[28] rubrica = {}
```

```
[29] rubrica
```

```
{}
```

```
[31] rubrica["Lorenzo"] = "345098"
     rubrica["Miriana"] = "333678"
```

```
rubrica
```

```
{'Lorenzo': '345098', 'Miriana': '333678'}
```

```
[38] rubrica["GianPio"] = "325298"
     rubrica["Nicole"] = "332628"
```

```
[39] rubrica.items()
```

```
dict_items([('GianPio', '325298'), ('Nicole', '332628')])
```

```
[42] for chiave, valore in rubrica.items():
       print(chiave+":"+valore)
```

```
GianPio:325298
Nicole:332628
```

```
for chiave, valore in rubrica:
  print(chiave+":"+valore)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-43-813750bf24f3> in <module>
----> 1 for chiave, valore in rubrica:
      2   print(chiave+":"+valore)

ValueError: too many values to unpack (expected 2)
```

SEARCH STACK OVERFLOW

Sets

- Set: object that stores a collection of data in same way as mathematical set
  - All items must be unique
  - Set is unordered
  - Elements can be of different data types

```
[44] myset = set()
```

```
[45] myset
```

```
set()
```

```
[46] altro_set = set(['a','b','c'])
```

```
[47] altro_set
```

```
{'a', 'b', 'c'}
```

```
[48] altro_ancora = set('abbccc')
```

```
altro_ancora
```

```
{'a', 'b', 'c'}
```

```
[1]  myset = {}

     print(type(myset))
```

```
     <class 'dict'>
```

```
[5]  myset = set()

     print(type(myset))
```

```
     <class 'set'>
```

```
     myset = {1}

     print(type(myset))
```

```
     <class 'set'>
```

Esercizio

Come posso creare un set contenente i 3 elementi "uno", "due e "tre"?

```
[3] myset = {'uno','due','tre'}
    myset
```

```
{'due', 'tre', 'uno'}
```

```
[4] myset = set('uno','due','tre')
    myset
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-97f3ddefb53a> in <module>
----> 1 myset = set('uno','due','tre')
      2 myset

TypeError: set expected at most 1 argument, got 3
```

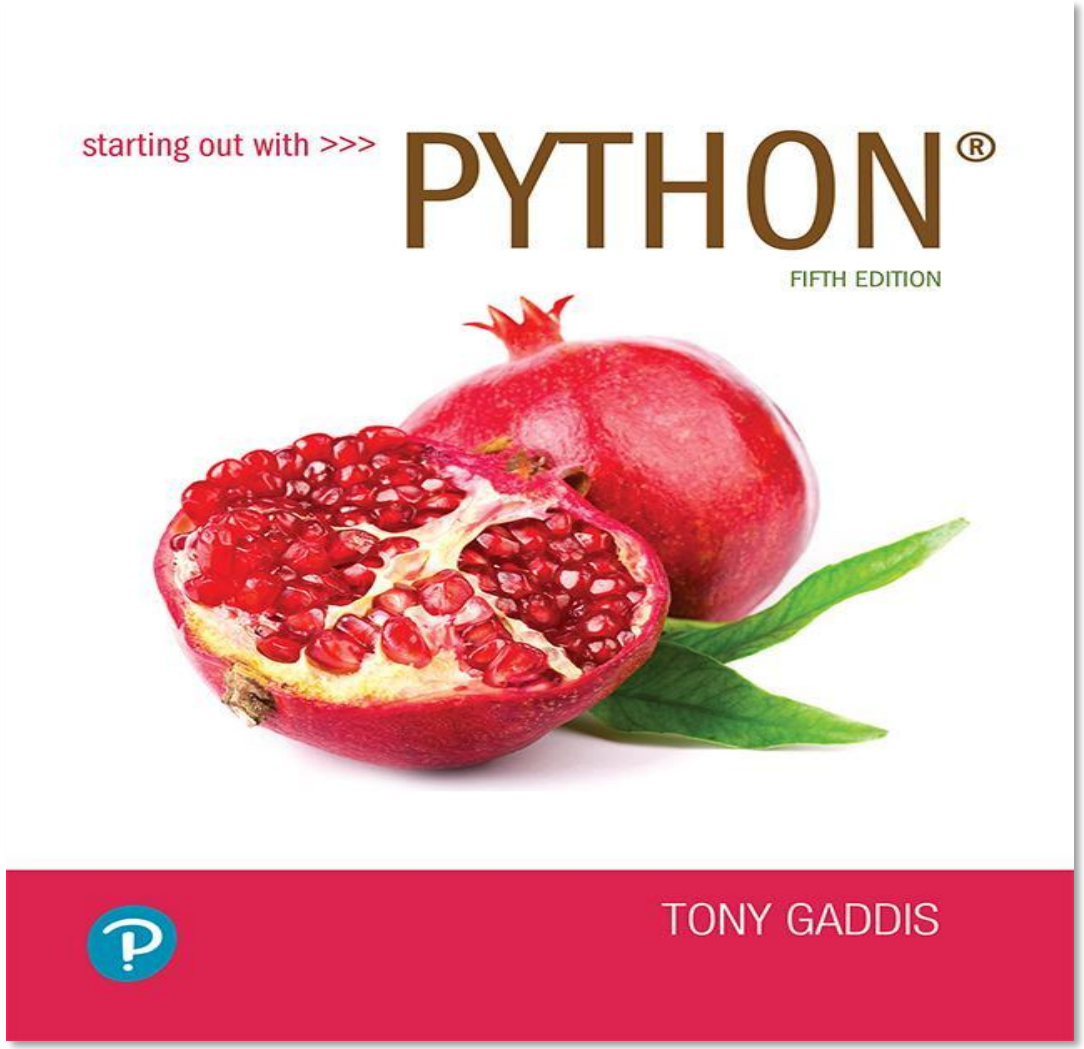
SEARCH STACK OVERFLOW

```
myset = set(['uno','due','tre'])
myset
```

```
{'due', 'tre', 'uno'}
```

Starting out with Python

Fifth Edition



Chapter 10

Classes and Object-Oriented Programming

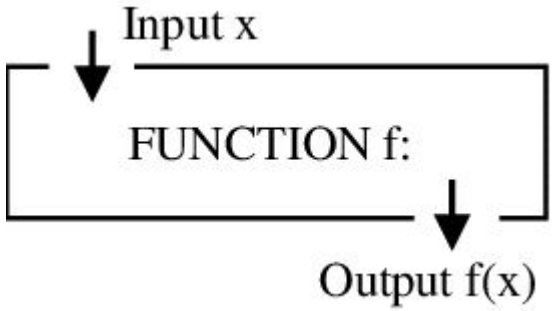
Topics

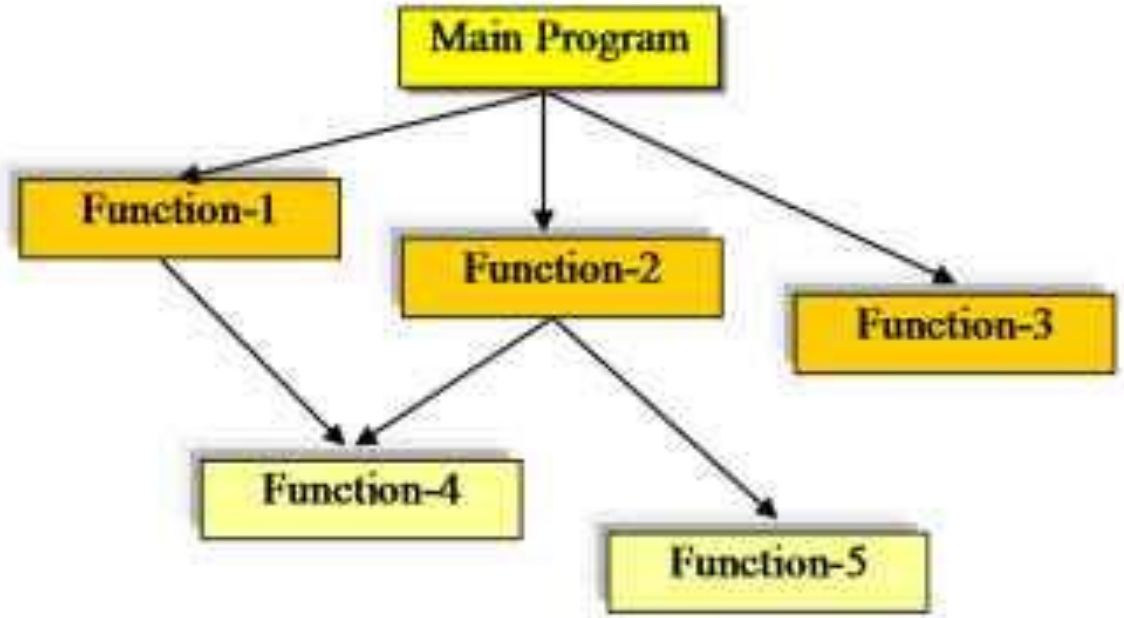
- Procedural and Object-Oriented Programming
- Classes
- Working with Instances
- Techniques for Designing Classes

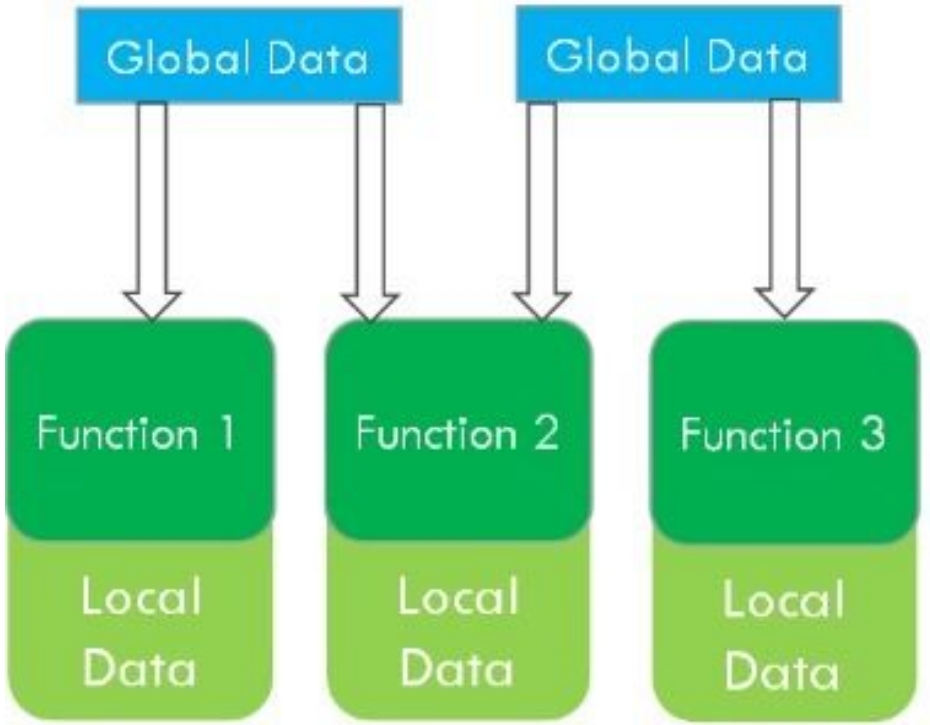
Procedural Programming

- Procedural programming: writing programs made of functions that perform specific tasks
  - Procedures typically operate on data items that are separate from the procedures
  - Data items commonly passed from one procedure to another
  - Focus: to create procedures that operate on the program's data

Procedural Programming



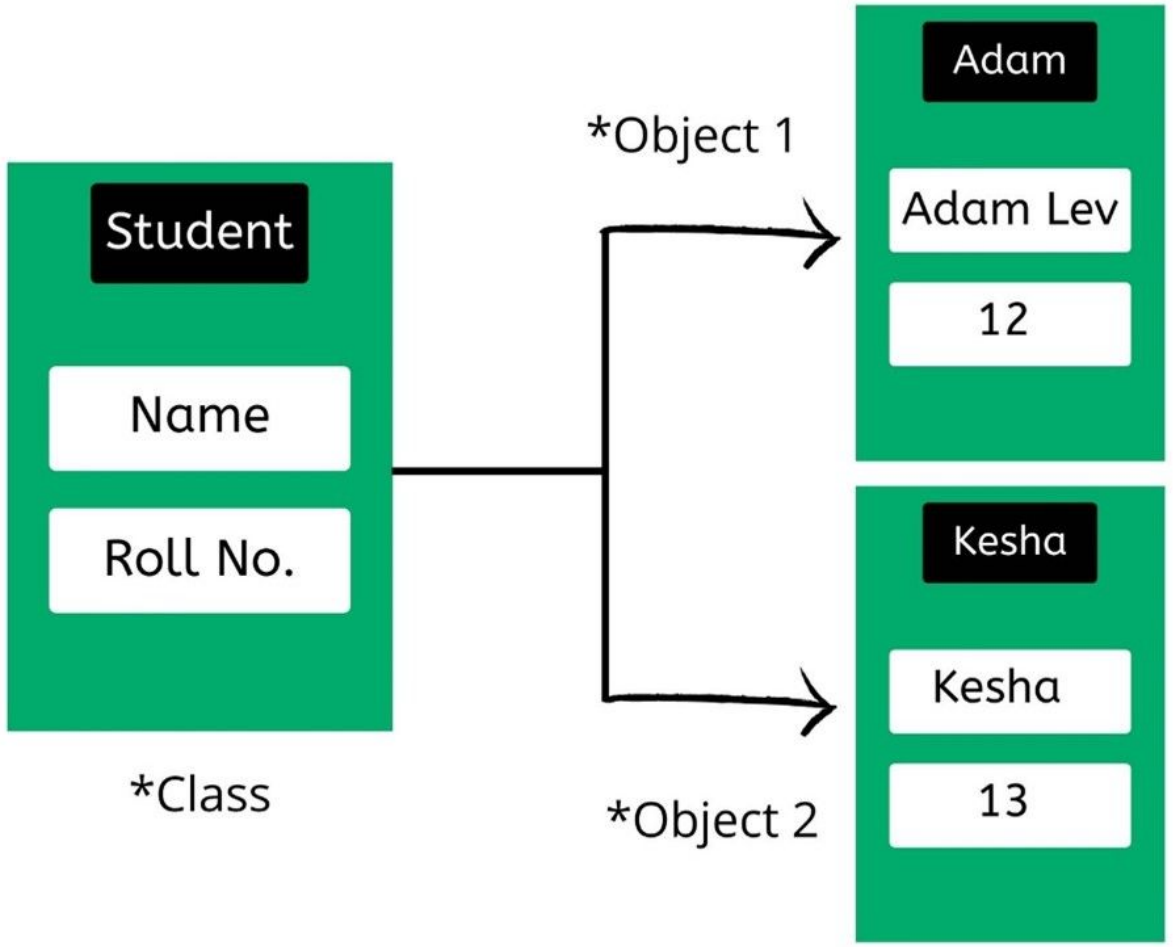




Object-Oriented Programming (1 of 4)

- Object-oriented programming: focused on creating objects
- Object: entity that contains data and procedures
  - Data is known as data attributes and procedures are known as methods
    - Methods perform operations on the data attributes
- Encapsulation: combining data and code into a single object

Object-Oriented Programming (1 of 4)



https://positiwise.com/blog/object-oriented-programming-vs-functional-programming-comparison/

Object-Oriented Programming (2 of 4)

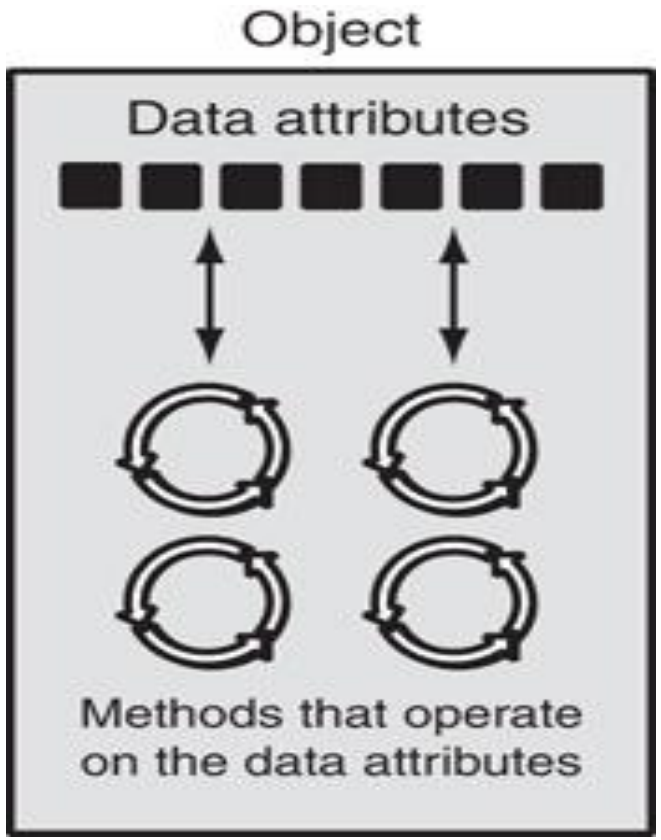


Figure 10-1 An object contains data attributes and methods

Object-Oriented Programming (3 of 4)

- Data hiding: object's data attributes are hidden from code outside the object
  - Access restricted to the object's methods
    - Protects from accidental corruption
    - Outside code does not need to know internal structure of the object
- Object reusability: the same object can be used in different programs
  - Example: 3D image object can be used for architecture and game programming

Object-Oriented Programming (4 of 4)

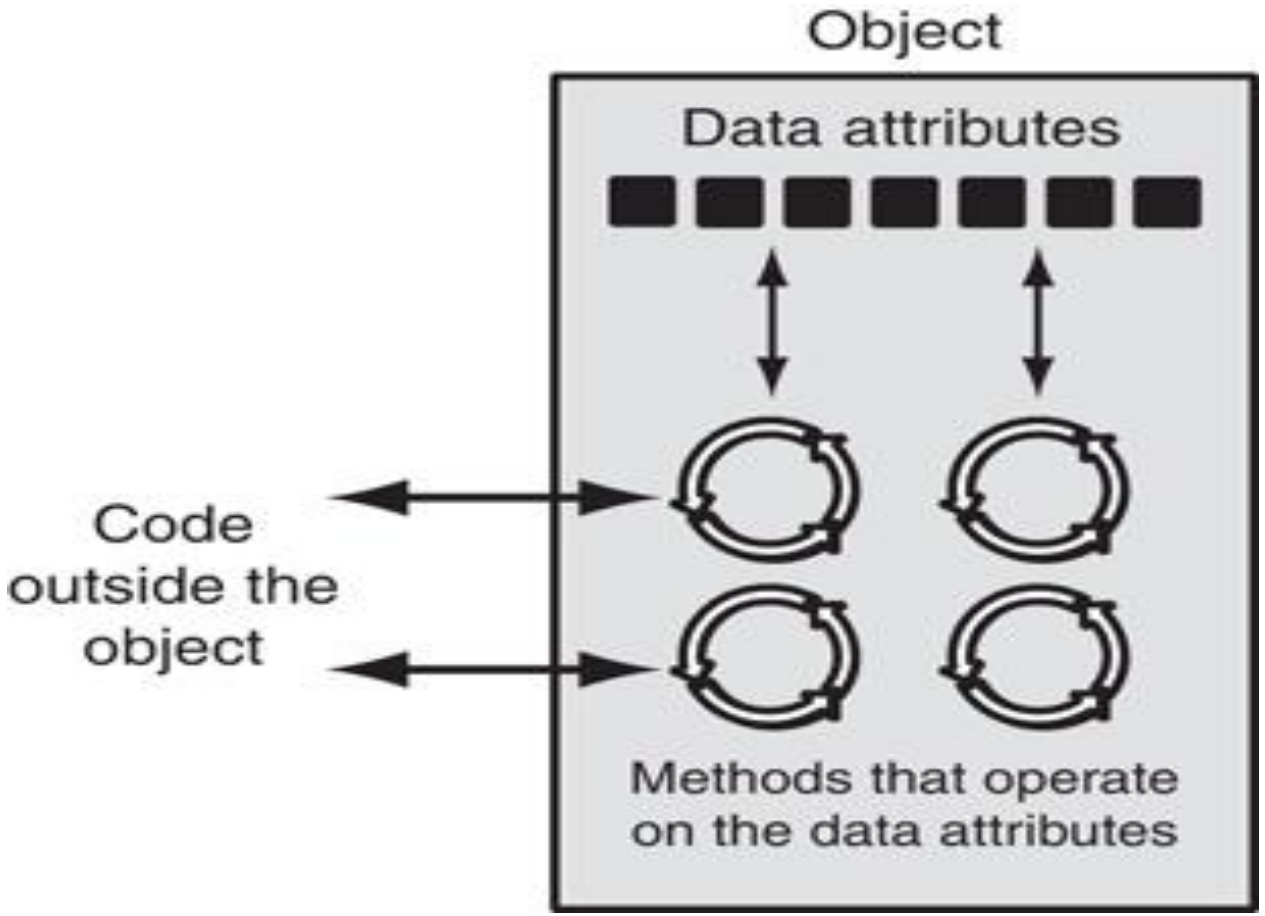
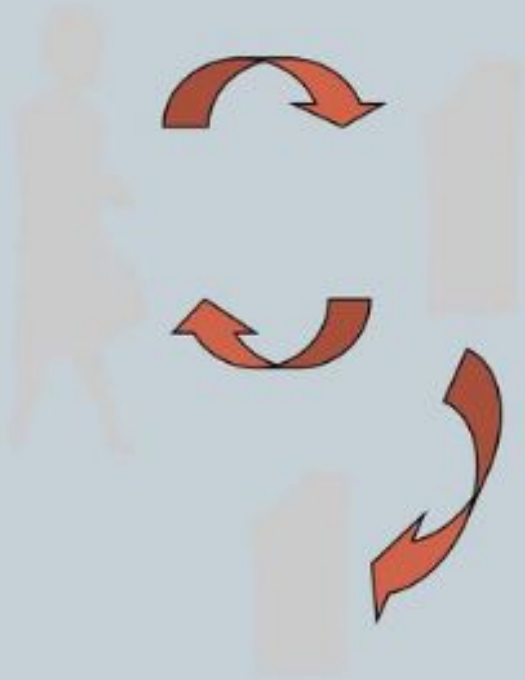


Figure 10-2 Code outside the object interacts with the object's methods

Procedural vs. Object-Oriented

- Procedural



Withdraw, deposit, transfer

- Object Oriented

Customer, money, account

An Everyday Example of an Object

- Data attributes: define the state of an object
  - Example: clock object would have `second`, `minute`, and `hour` data attributes
- Public methods: allow external code to manipulate the object
  - Example: `set_time, set_alarm_time`
- Private methods: used for object's inner workings

Classes (1 of 3)

- Class: code that specifies the data attributes and methods of a particular type of object
  - Similar to a blueprint of a house or a cookie cutter
- Instance: an object created from a class
  - Similar to a specific house built according to the blueprint or a specific cookie
  - There can be many instances of one class

Classes (2 of 3)

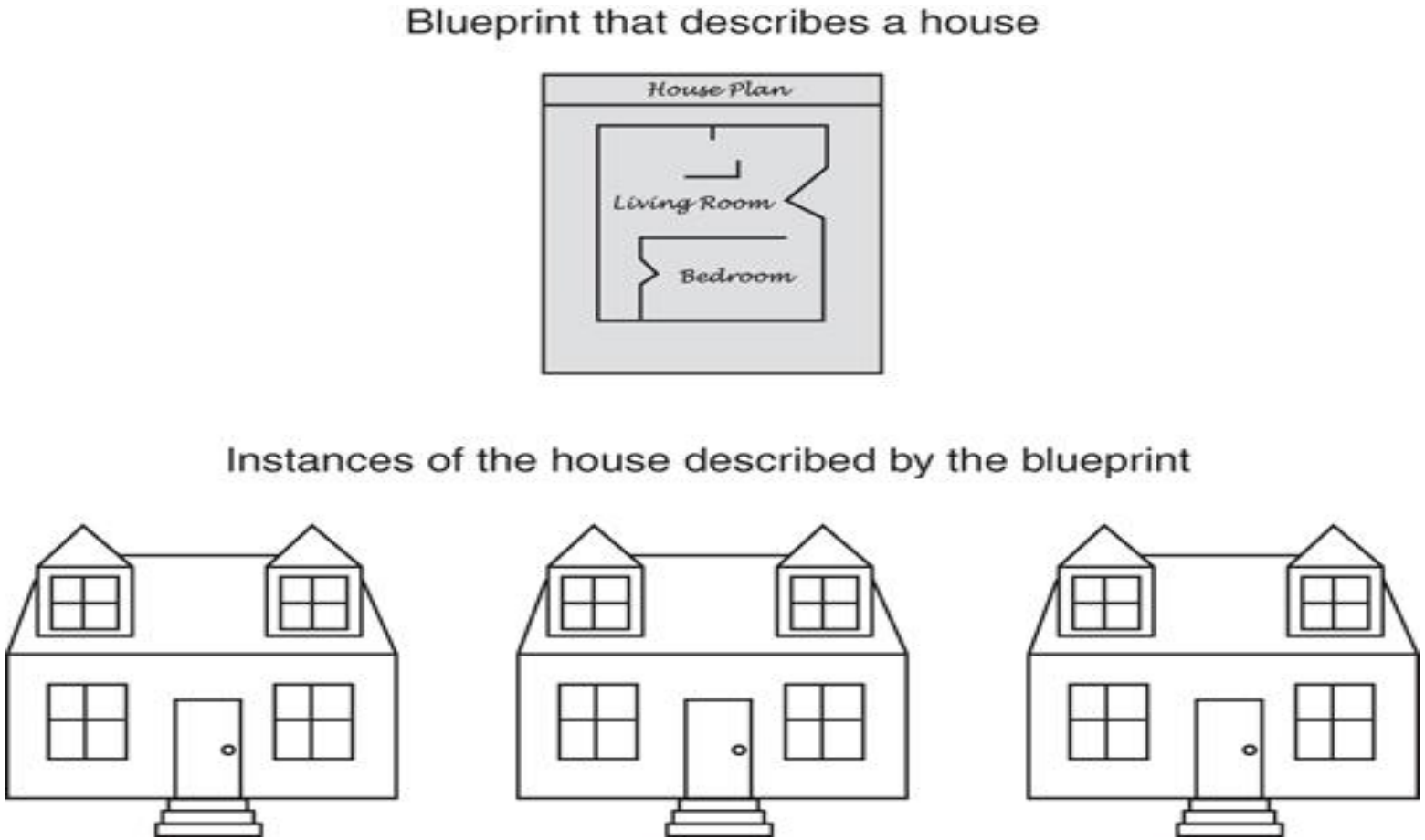
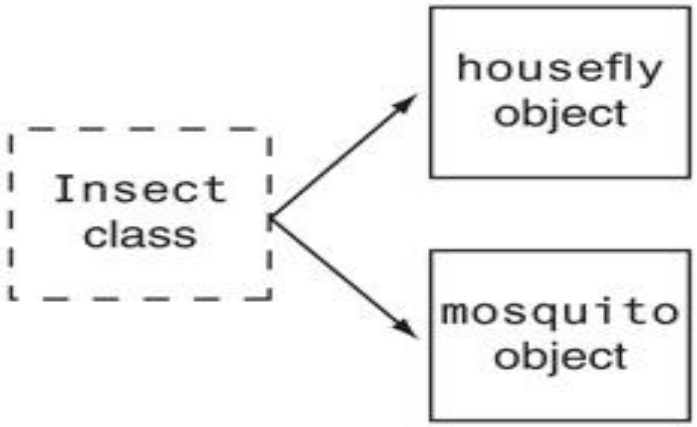


Figure 10-3 A blueprint and houses built from the blueprint

Classes (2 of 3)

Classes (3 of 3)

The Insect class describes the data attributes and methods that a particular type of object may have.



The housefly object is an instance of the Insect class. It has the data attributes and methods described by the Insect class.

The mosquito object is an instance of the Insect class. It has the data attributes and methods described by the Insect class.

Figure 10-4 The `housefly` and `mosquito` objects are instances of the `Insect` class

Class Definitions (1 of 4)

- Class definition: set of statements that define a class's methods and data attributes
  - Format: begin with `class Class_name:`
    - Class names often start with uppercase letter
  - Method definition like any other python function definition
    - `self` parameter: required in every method in the class – references the specific object that the method is working on

Class Definitions (2 of 4)

- Initializer method: automatically executed when an instance of the class is created
  - Initializes object's data attributes and assigns `self` parameter to the object that was just created
  - Format: `def __init__ (self):`
  - Usually the first method in a class definition

```
import random

class Coin:

  def __init__(self):
    self.sideup = 'Heads'

  def toss(self):
    if random.randint(0, 1) == 0:
      self.sideup = 'Heads'
    else:
      self.sideup = 'Tails'

  def get_sideup(self):
    return self.sideup
```

Class Definitions (3 of 4)

1. An object is created in memory from the `Coin` class.

2. The `Coin` class's `__init__` method is called, and the `self` parameter is set to the newly created object

After these steps take place, a `Coin` object will exist with its `sideup` attribute set to `'Heads'`.

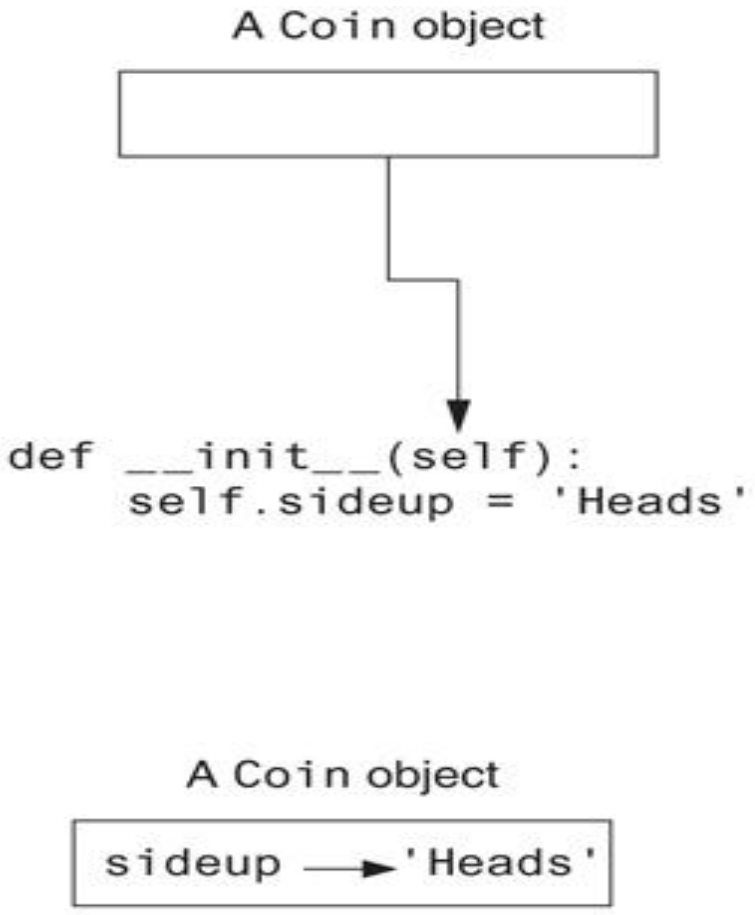


Figure 10-5 Actions caused by the `Coin()` expression

Class Definitions (4 of 4)

- To create a new instance of a class call the initializer method
  - Format: *My_instance = Class_Name*()
- To call any of the class methods using the created instance, use dot notation
  - Format: *My_instance*.*method*()
  - Because the `self` parameter references the specific instance of the object, the method will affect this instance
    - Reference to `self` is passed automatically

```
def main():

  my_coin = Coin()

  print('This side is up:', my_coin.get_sideup())

  print('I am tossing the coin ...')
  my_coin.toss()

  print('This side is up:', my_coin.get_sideup())


main()
```

```
This side is up: Heads
I am tossing the coin ...
This side is up: Heads
```

```
def main():

  my_coin = Coin()

  print('This side is up:', my_coin.get_sideup())

  print('I am tossing the coin ...')
  my_coin.toss()

  # But now I'm going to cheat! I'm going to
  # directly change the value of the object's
  # sideup attribute to 'Heads'.
  my_coin.sideup = 'Heads'

  print('This side is up:', my_coin.get_sideup())


main()
```

```
This side is up: Heads
I am tossing the coin ...
This side is up: Heads
```

Hiding Attributes and Storing Classes in Modules

- An object's data attributes should be private
  - To make sure of this, place two underscores (__) in front of attribute name
    - Example: `__current_minute`
- Classes can be stored in modules
  - Filename for module must end in .py
  - Module can be imported to programs that use the class

```
[20] import random

     class Coin:

       def __init__(self):
         self.__sideup = 'Heads'

       def toss(self):
         if random.randint(0, 1) == 0:
           self.__sideup = 'Heads'
         else:
           self.__sideup = 'Tails'

       def get_sideup(self):
         return self.__sideup

       def set_sideup(self,value):
         self.__sideup = value
```

```
my_coin = Coin()
print(my_coin.get_sideup())
my_coin.set_sideup('Tails')
print(my_coin.get_sideup())
```

```
Heads
Tails
```

```
def main():
    my_coin = Coin()

    print('This side is up:', my_coin.get_sideup())

    print('I am going to toss the coin 100 times:')
    for count in range(100):
        my_coin.toss()
        print(my_coin.get_sideup())

main()
```

The `__str__` method

- Object's state: the values of the object's attribute at a given moment
- `__str__ method`: displays the object's state
  - Automatically called when the object is passed as an argument to the `print` function
  - Automatically called when the object is passed as an argument to the `str` function

Working With Instances (1 of 3)

- Instance attribute: belongs to a specific instance of a class
  - Created when a method uses the `self` parameter to create an attribute
- If many instances of a class are created, each would have its own set of attributes

Working With Instances (2 of 3)

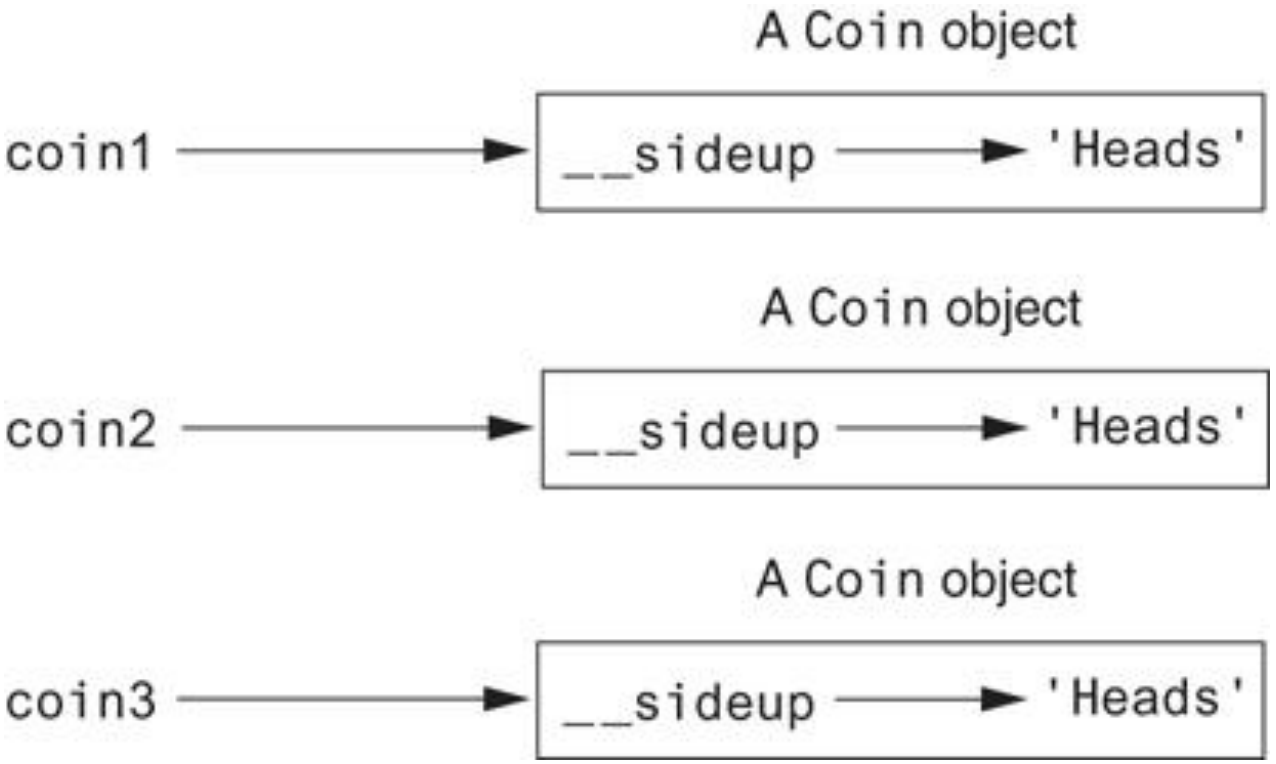


Figure 10-7 The `coin1`, `coin2`, and `coin3` variables reference three `Coin` objects

Working With Instances (3 of 3)

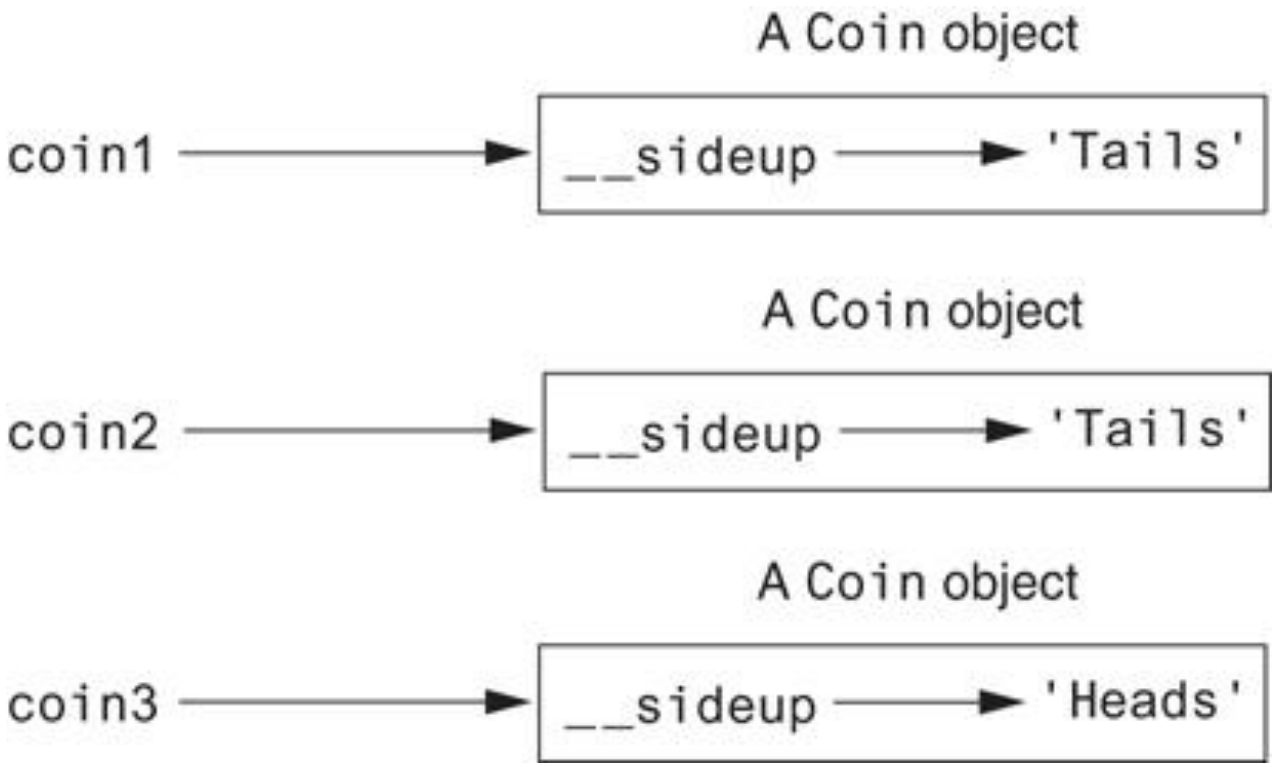


Figure 10-8 The objects after the `toss` method

Accessor and Mutator Methods

- Typically, all of a class's data attributes are private and provide methods to access and change them
- Accessor methods: return a value from a class's attribute without changing it
  - Safe way for code outside the class to retrieve the value of attributes
- Mutator methods: store or change the value of a data attribute

Passing Objects as Arguments

- Methods and functions often need to accept objects as arguments
- When you pass an object as an argument, you are actually passing a reference to the object
  - The receiving method or function has access to the actual object
    - Methods of the object can be called within the receiving function or method, and data attributes may be changed using mutator methods

Techniques for Designing Classes (1 of 3)

- UML diagram: standard diagrams for graphically depicting object-oriented systems
  - Stands for Unified Modeling Language
- General layout: box divided into three sections:
  - Top section: name of the class
  - Middle section: list of data attributes
  - Bottom section: list of class methods

Techniques for Designing Classes (2 of 3)

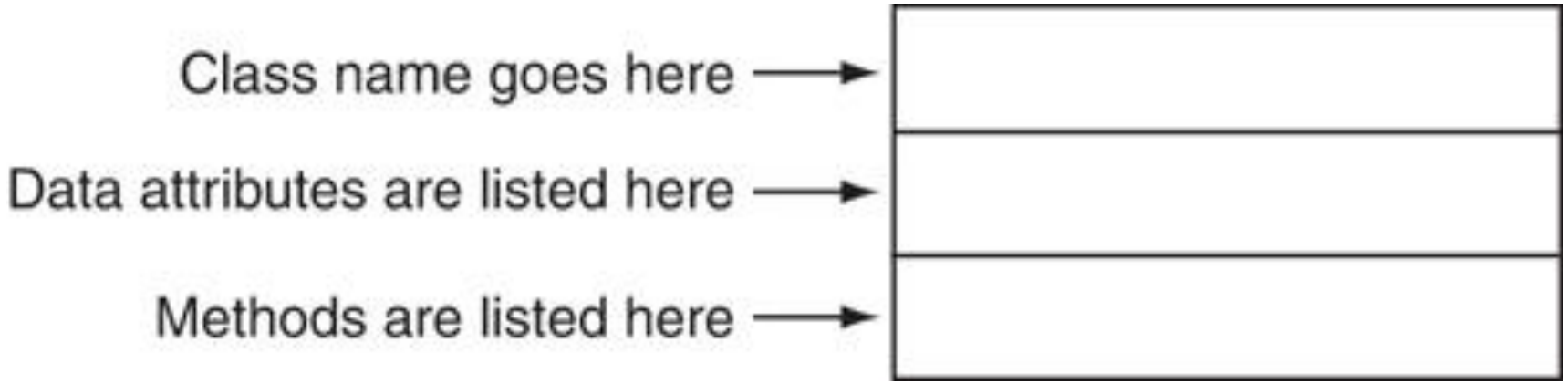


Figure 10-9 General layout of a UML diagram for a class

Techniques for Designing Classes (3 of 3)

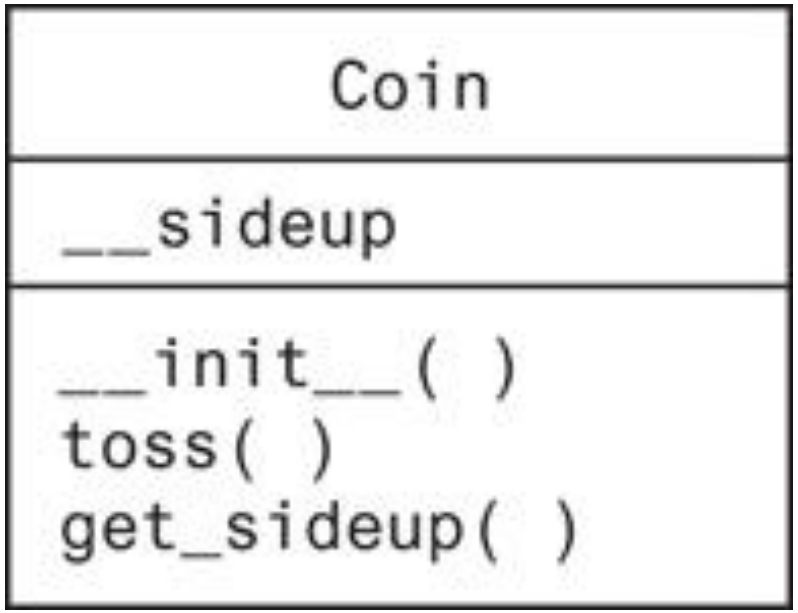


Figure 10-10 UML diagram for the Coin class

Finding the Classes in a Problem (1 of 4)

- When developing object oriented program, first goal is to identify classes
  - Typically involves identifying the real-world objects that are in the problem
  - Technique for identifying classes:
    1. Get written description of the problem domain
    2. Identify all nouns in the description, each of which is a potential class
    3. Refine the list to include only classes that are relevant to the problem

Finding the Classes in a Problem (2 of 4)

1. Get written description of the problem domain
   - May be written by you or by an expert
   - Should include any or all of the following:
     - Physical objects simulated by the program
     - The role played by a person
     - The result of a business event
     - Recordkeeping items

Finding the Classes in a Problem (3 of 4)

2. Identify all nouns in the description, each of which is a potential class
   - Should include noun phrases and pronouns
   - Some nouns may appear twice

Finding the Classes in a Problem (4 of 4)

3. Refine the list to include only classes that are relevant to the problem
   - Remove nouns that mean the same thing
   - Remove nouns that represent items that the program does not need to be concerned with
   - Remove nouns that represent objects, not classes
   - Remove nouns that represent simple values that can be assigned to a variable

Identifying a Class's Responsibilities

- A classes responsibilities are:
  - The things the class is responsible for knowing
    - Identifying these helps identify the class's data attributes
  - The actions the class is responsible for doing
    - Identifying these helps identify the class's methods
- To find out a class's responsibilities look at the problem domain
  - Deduce required information and actions

Summary

- This chapter covered:
  - Procedural vs. object-oriented programming
  - Classes and instances
  - Class definitions, including:
    - The `self` parameter
    - Data attributes and methods
    - `__init__` and `__str__` functions
    - Hiding attributes from code outside a class
  - Storing classes in modules
  - Designing classes

Corso di STATISTICA, INFORMATICA, ELABORAZIONE DELLE INFORMAZIONI

UNIVERSITÀ DEGLI STUDI DELLA BASILICATA

Modulo di Sistemi di Elaborazione delle Informazioni

Classi

Docente:
Domenico Daniele Bloisi